2003－HPC－96　(5)
2003／10／16

ＬＩＳＴＶＥＣ指示行を使った粒子シミュレーション (メモリーを節約し、かつ高速化を可能にする手法)

杉山　徹[?]　寺田直樹[??]　村田健史[???]
大村善治[?]　臼井英之[?]　松本　紘[?]

宇宙空間プラズマを粒子的に扱うシミュレーションを行う場合は、主にPIC (Particle-In-Cell) 法と呼ばれる手法を用いて実行される。すなわち、場の量（電流、電磁場など）を空間格子点上で定義し、プラズマ粒子を空間にランダムに配置する。その中で、粒子の運動と場の量を結びつけるために粒子の速度のモーメントを計算するが、この計算をベクトル化して行うには、粒子がランダムに分布しているため工夫を要する。ここでは、従来の方法と、地球シミュレーター等に使われているSXベクトル計算機で使用可能なコンパイラ指示オプションを用いた方法とを比較した結果を報告する。なお、実際のアルゴリズムは、NECが保持する特許事項である。

Vectorized Particle Simulation using “LISTVEC” compile-directive on SX super-computer

TOORU SUGIYAMA[?], NAOKI TERADA[??], TAKESHI MURATA[???]
YOSHIHARU OMURA[?], HIDEYUKI USUI[?], HIROSHI MATSUMOTO[?],

PIC (Particle-In-Cell) method is frequently used for space plasma particle simulations. In this method, the field components (i.e. current, electric/magnetic field) are defined on grids and plasma particles are randomly distributed in space. To obtain the current, we need to calculate the velocity moment of the particles. The calculation is hard to perform on vector-type computers. Here, we introduce a new method where “LISTVEC” compile-directive on SX super-computer is used. We compare it with the conventional robust method.

1. はじめに

宇宙空間プラズマを対象としたシミュレーションの１つとして、粒子法と呼ばれる手法がある。そこでは、プラズマ粒子の運動論効果を解明するために、プラズマ粒子1つ1つの運動と、周囲の電磁場との相互作用を self-consistent に計算する。粒子的に行うシミュレーションとしては、天文の分野で行われている多体問題のシミュレーション GRAPE（物体と重力場）があるが、宇宙空間プラズマに関しては、そのような手法はあまり用いられず、代わりに、PIC (Particle-In-Cell) と呼ばれる手法が使われる。つまり、場の量（電流、電磁場など）を空間格子点上でのみ定義し、プラズマ粒子を空間にランダムに配置し、粒子のモーメント値（電荷密度、電流密度）は、粒子位置に隣接する格子点にのみ配分する。ここで問題となるのが、ランダムな位置に分布する粒子データ

[?] 京都大学宙空電波科学研究センター
Radio Science Center for Space and Atmosphere (RASC), Kyoto University
[??] 名古屋大学太陽地球環境研究所
Solar-Terrestrial Environment Laboratory (STEL), Nagoya University
[???] 愛媛大学総合情報メディアセンター
Center for Information Technology, Ehime University

に対し、モーメント値をベクトル計算機でどのようにベクトル計算を行うかである。後述のような従来の方法で行うと、最大限のベクトル長で、99%を超えるベクトル化率で計算が可能であるが、作業用のメモリーを大量に使うため、実行するシミュレーションの規模が制限される。本論文では、新たな方法を紹介し、従来の方法と比較した結果を紹介する。実際にプラズマ粒子計算を行ったコードは、イオンのみを粒子として扱い、電子を電荷中性を満たす慣性の無い流体として扱うHYBRID コードに対して検証した。また使用したベクトル計算機は、京都大学宙空電波科学研究センターで運用されているNEC 社製の SX－5である。

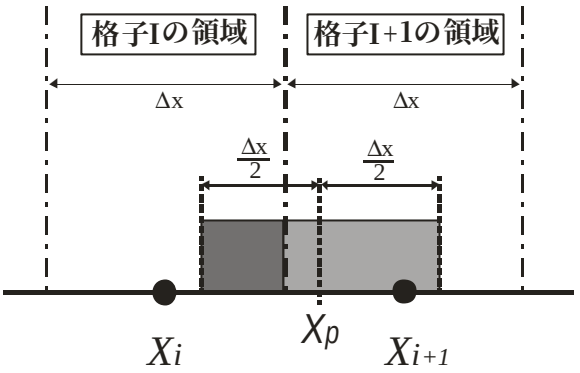

図１:粒子の形状と密度配分の関係。粒子の密度は格子領域に占める粒子の大きさに比例して分配される。(注:粒子の形状は、長方形以外にも三角形やスプライン関数型などがあり、分配される格子の数も2つ以上となるものもある。)

2　粒子モーメントの計算法

粒子法では、位相空間での分布関数が十分滑らかでなければ、正しい運動論を議論することができない。そのためには、格子点あたりの粒子数が数百個以上であることが望ましい。そのため、粒子法の計算では、格子点上の電磁場に対する計算量（流体的な計算）に対し、粒子に関する計算量が粒子数に応じて大きくなる。このことから、計算効率の向上には、粒子計算部を効率よく計算する必要がある。

粒子計算部では、超粒子として扱われる図1にあるような格子間隔と同じ長さの長方形の粒子に対し、以下の2つの計算が行われる（簡単のため説明は1次元モデルで行う）。(1)運動方程式から、粒子の速度と位置を更新する。ここでは、格子上(Xi)で定義されている電磁場の値を、粒子の位置 (Xp) に内挿した値が用いられる。この計算では、特に工夫することなくベクトル化された計算が行われる。(2)粒子の速度と位置から、電荷密度と電流密度を計算する。＜プログラム1＞にあるようなループで密度計算を行う場合、粒子の背番号Nと粒子の位置がランダムなため、Nに対し格子点の位置 I の依存関係が不明となり、この DO ループはベクトル計算ができない。すなわち図2にあるように、粒子背番号2と3、6と7が

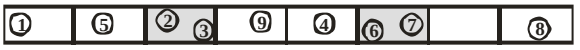

図２：ベクトル化を阻止する粒子の分布例。配列 DNS の箱に粒子が分布している概念図。

＜プログラム1＞

```
!   N: 粒子の背番号,   Nmax:   総粒子数
!   DNS:   密度
do 100 N = 1, Nmax
   I = INT( x(N) )
    S1 = Shape_Func1( x(N) )
   S2 = Shape_Func2( x(N) )
    DNS(  I) = DNS(  I) + S1
   DNS(I+1) = DNS(I+1) + S2
100 continue
```

同じ格子間に存在している場合は、配列DNSがベクトル計算できない。そこで考え出されたベクトル化の方法は、作業用配列 WRK_*(1:Gmax,1:Lv) を密度成分の数だけ用意し、＜プログラム2＞にあるように、内側の200番 DO ループでベクトル計算を行う。その概念図を図3に示す。この方法では、たとえ図3に

＜プログラム2＞

```
real   X(N2max,Lv), V(N2max,Lv)
real   WRK_N(Gmax,Lv), DNS(Gmax)
real   WRK_V(Gmax,Lv), FLX(Gmax)
do 100 N = 1, N2max
do 200 L = 1, Lv
     S1 = Shape_Func1( X(N,L) )
    S2 = Shape_Func2( X(N,L) )
    WRK_N(  I,L) = WRK_N(  I,L) + S1
     WRK_N(I+1,L) = WRK_N(I+1,L) + S2
    WRK_V(  I,L) = WRK_V(  I,L) + S1 * V(N,L)
     WRK_V(I+1,L) = WRK_V(I+1,L) + S2 * V(N,L)
200 continue
100 continue
do 300 I = 1, Gmax
     DNS(I) = SUM( WRK_N(I,1:Lv) )
     FLX(I) = SUM( WRK_V(I,1:Lv) )
300 continue
```

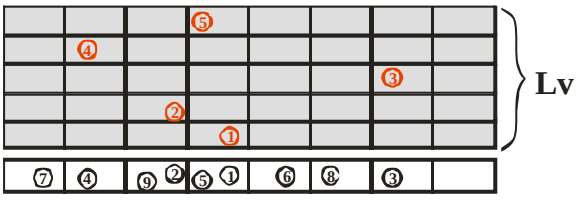

図３：ベクトル化を可能にした密度計算方法の概念図。(格子点数×Lv) というサイズの作業用2次元配列のためのメモリー量が必要。

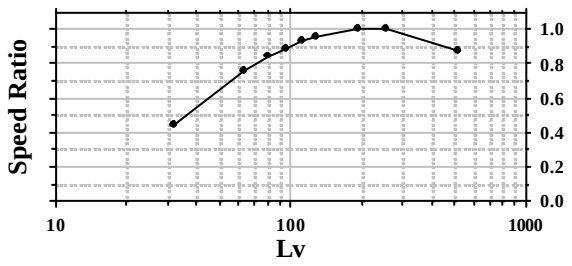

図４：Lv に対する計算速度依存。Lv=256 での速度で規格化してある。

あるように、粒子背番号1と5が同じ格子間に存在しても、ループ変数Lの値が異なるためベクトル計算が可能である。Lv の大きさが、ベクトル計算のベクトル長となるため、256 の時、すなわちシステムの最長ベクトル長と同じ大きさの時に計算速度が最速となる。Lv に対する計算速度の依存性を図4に示す（後述の RUN1 による測定）。しかし、この方法では、作業用配列分のメモリーが余計に消費されるため、シミュレーションの規模に制限を与えてしまう。Lv を 256 にすることは、格子点あたり 256 個の粒子を分布させることと同じであり、いかに作業用配列のサイズが大きいかがわかる。実際の計算では、4種類（位置と速度 3成分）の作業用配列が必要となり、それだけで、格子点あたり 粒子 256×4=1024 個分のメモリーを消費することとなる。

3　メモリーを節約したモーメント計算

3.1　作業用配列の再利用

作業用配列を1つだけ用意し、＜プログラム2＞の 100 番と200 番のループを、位置と速度 3成分についてそれぞれ行い、密度を求めればメモリーを節約することができる。表1に、実行速度がどの程度変わるかを測定した2次元計算の結果を示す。従来の方法で行った計算を RUN1、再利用し4度ループをまわした計算を RUN2 とする。

格子点数　　X方向　512　　Y方向　512
粒子数　　格子点あたり 128　個

	RUN 1	RUN 2
メモリーサイズ	3.65 GB	2.04 GB
100 ステップ	475.7	586.3
ベクトル化率	99.7	99.5
2000 ステップ	9473.6	11541.5
ベクトル化率	99.7	99.5

表1：計算に要した時間（秒）とベクトル化率（%）

RUN2 では、4倍多くループをまわしているが、約1.2 倍の時間増で終わるため、メモリー量と速度の比を考えると、有用な方法の1つである。

３２　コンパイル指示行 LISTVEC の利用

次に、本論文の主となるコンパイル指示行によるベクトル化の結果を示す。図 2にあるように、同じ配列に背番号の近い粒子が入った場合、ベクトル計算は行われないが、入らない場合はベクトル計算が可能である。よって、密度計算のループを粒子の背番号でまわしている時に、ベクトル計算が可能な期間はベクトル計算をし、衝突が生じた時だけ補正を行えば、作業配列を用意しなくてもベクトル計算が可能となる。そのような計算を実行させるコンパイル指示行（"LISTVEC"）が、SXのコンパイラに用意されており、使用例を＜プログラム３＞に示す。

LISTVEC 指示行を付けた DO ループでは、I番目の密度配列に加算する時に依存があるかを実行時に判断し、依存がある場合には、それを補正する処理が挿入されている。したがって、依存がある要素が少ない場合には高速に実行でき、依存がある要素が多い場合には、補正のためのオーバヘッドが大きくなり、通常のスカラー計算より遅くなる場合がある。この方法の良い所は、依存のチェックに、マスタレジスタを利用しているため、余分にメモリーを消費することが無いという点である。表 2に、実行速度がどの程度変わるかを測定した 2次元計算の結果を示す。本計算を RUN３とする。また、比較のため＜プログラム1＞でスカラー実行させた計算 RUN4 の結果も示す。RUN3 は、RUN1 に対して、約 1.57 倍、RUN2に対しては、約 1.29 倍の時間を要したが、使用するメモリー量が、それぞれ 2.4 倍、1.33 倍小さいことを考えれば、実用可能な方法と考えられる。

＜プログラム3＞

```
!CDIR LISTVEC
do 100 N = 1, Nmax
   I = INT( x(N) )
    S1 = Shape_Func1( x(N) )
   DNS(  I) = DNS(  I) + S1
   FLX(  I) = FLX(  I) + S1 * V(N)
100 continue
!CDIR LISTVEC
do 110 N = 1, Nmax
   I = INT( x(N) )
   S2 = Shape_Func2( x(N) )
   DNS(I+1) = DNS(I+1) + S2
    FLX(I+1) = FLX(I+1) + S1 * V(N)
110 continue
```

	RUN 3	RUN 4
メモリーサイズ	1.50 GB	1.50 GB
100 ステップ	753.7	145109.0
ベクトル化率	99.7	0
2000 ステップ	14888.5	291891.9
ベクトル化率	99.7	0

表 2 :計算に要した時間（秒）とベクトル化率（%）

３３　コンパイル指示行 LISTVEC の使用制限

LISTVECを使用したベクトル化では、現在までに、我々の使用法の下で、以下の 3つの使用制限があることがわかった。

(1) 補正作業が必要となるため、1つのループの中で同じ名前の配列を複数回使用することはできない。
(2) 補正作業を必要とする依存関係の発生を確率的に下げるため、格子点の数は、最大ベクトル長の 256 より十分大きくなければならない。
(3) 補正作業の発生確率を下げるために、粒子を空間にランダムに分布させなければならない。

＜プログラム 3＞において、ループを2つに分けている理由は、上記制限 (1) のためである。また、配列名が異なるため、電荷密度と電流密度を同じループで計算することが可能である。

一方、配列名がことなれば良いということから、110番ループで使われる配列の名前を変え、100番ループの中で計算することも可能である (RUN5)。その例を＜プログラム 4＞に示す。この方法では、粒子のモーメント値を分配する格子数分のメモリーが消費されるが、配分される格子点数は、1、2、3 次元計算で、それぞれ、2つ、4つ、9つ、であるため、従来の方法に比べ消費されるメモリー量は少ない。表3 に、実行速度がどの程度変わるかを測定した 2次元計算の結果を示す。特記すべきことは、本方法で行うと、**RUN2 よりも高速に計算できる**ことである。また、RUN1 に対しても1.15 倍の時間で実行できる。

＜プログラム 4＞

```
!CDIR LISTVEC
do 100 N = 1, Nmax
   I = INT( x(N) )
    S1 = Shape_Func1( x(N) )
    S2 = Shape_Func2( x(N) )
    DNS(  I)  = DNS(  I)  + S1
   DNS_2(I+1) = DNS_2(I+1)  + S2
    FLX(  I)  = FLX(  I)  + S1 * V(N)
    FLX_2(I+1) = FLX_2(I+1) + S2 * V(N)
100 continue
do 300 I = 1, Gmax
    DNS(I) = DNS(I) + DNS_2(I)
    FLX(I) = FLX(I) + FLX_2(I)
300 continue
```

	RUN 3	RUN 5
メモリーサイズ	1.50 GB	1.53 GB
100 ステップ	753.7	547.1
ベクトル化率	99.7	99.8
2000 ステップ	14888.5	10907.8
ベクトル化率	99.7	99.8

表 3 :計算に要した時間 (秒) とベクトル化率 (%)

次に、制限 (2) に関して、格子点数に対する計算速度を測定した結果を表 4に示す。システムサイズに 16 倍の差があるにもかかわらず、計算時間が 9.8 倍であることから、補正する確率が下がった効果が現れている。

格子点数	メモリーサイズ	RUN 3
128×128	189 MB	1514.9
512×512	1.50 GB	14888.5

表 4 :計算に要した時間 (秒)

初期状態として、粒子をシミュレーション空間に分布させる時に関する注意点として (3) が挙げられる。プログラミングを簡単にするためや、乱数発生を極力抑えたい場合、隣り合う背番号の粒子は近接した初期位置を持たせることがある。しかし、この方法では、補正作業を必要とする依存関係の発生が多発するため、実行速度がスカラー計算速度以下になる。今回の速度測定では、粒子の初期位置は乱数を発生させ空間にランダムに分布させている。その効果は、計算に要する時間がタイムステップ数に単純に比例することに見られる。(表 3)

4. 考 察

従来のベクトル化のための手法に比べ、格段にメモリー節約し、指示行を利用したベクトル手法を紹介した。1つのノード中のメモリーサイズを大きくせずノード数を多くして大規模なシミュレーションを行うシステムで、従来の方法でベクトル化を行うと、作業配列にメモリーを消費されてしまい、肝心の粒子の数が十分大きく取れない問題が発生する。実際にわれ

われは、地球シミュレーターにおいて100ノード以上使い1.6TB以上を使用した計算を行う計画であったが、1ノード内のメモリーが16GBしかなく、従来の方法では、実行不可能であることがわかった。しかし、本論文で紹介した方法を用いれば、十分な量の粒子を入れたシミュレーションが実行可能となり、また、計算に要する時間の伸びが1.15倍程度に抑えられることが判明した。この速度は、RUN1による計算で、Lv=96にした値と同程度であるため（図4）、格子点あたり約384 (=96×4) 個分の粒子に相当するメモリーを節約できたことを意味する。一方、作業用配列を使う場合には、全ての密度成分に対してメモリーを用意しなければ、LISTVEC指示行を用いた方法より高速に実行できない。このため、十分大きな共有メモリーをノード内に持たないベクトル計算機では、LISTVEC指示行を使う方が有効である。

また、余談であるが、ベクトル化率も、従来の方法と同じく99.7-99.8％を達成しているため、われわれの用いるコードは、地球シミュレーターを有効に利用できるコードであることが示された。